CSc 120
Introduction to Computer Programming II

CODE EXAMPLES 02
def best(list_of_values):
    max_avg = 0.0
    best_items = []

    for value in list_of_values:
        value_avg = compute_average(value)
        if value_avg > max_avg:
            # found a higher best score: re-initialize
            max_avg = value_avg
            best_items = [value]
        elif value_avg == max_avg:
            # found another item with highest score: add to the list
            best_items.append(value)
        else:
            # value_avg < max_avg: ignore and move on
            continue

    return best_items

Q: IS THIS GOOD STYLE?
continue: pros and cons

def foo(...):
    for ...
        if ... :
            ...
        elif ... :
            ...
    else:
        # do nothing
        continue

return

😊 skips to the next iteration (as intended)

😡 not exactly what a continue is intended for
   – “skip the rest of the loop body”

😡 what if, later, we introduced more code into the loop body?
   – then continue would be a bug

CONSENSUS: continue is not good style here
omit the `else` branch: pros and cons

```python
def foo(...):
    for ...
        if ... :
            ...
        elif ... :
            ...
    return
```

- **😊 most direct:** doesn't have any "do-nothing" code
- **😢 some people find the `elif` without an `else` visually weird**
  - related issue: someone reading the code might wonder whether the omission of the `else` is a feature or a bug

If you use this style, make sure you add a comment saying that the `else` is deliberately omitted (+ why)
pass: pros and cons

def foo(...):
    for ...
        if ...
            ...
    elif ...
        ...
    else:
        # do nothing
            pass
    return

😊 does nothing (as intended)
😊 does not interfere with the loop

CONSENSUS: pass is good style here
Some code from assg 2 (Pokemon)
avg_dict = {}
for line in mydict.keys():
    for n in range(7):
        sum = 0
count = 0
for line2 in mydict[line].keys():
    temp = mydict[line][line2]
    sum += int(temp[n])
count += 1

Q: IS THIS GOOD STYLE?
avg_dict = {}
for line in mydict.keys():
    for n in range(7):
        sum = 0
        count = 0
        for line2 in mydict[line].keys():
            temp = mydict[line][line2]
            sum += int(temp[n])
            count += 1

Q: Are these names descriptive?
avg_dict = {}  
for line in mydict.keys():  
   for n in range(7):  
      sum = 0  
      count = 0  
      for line2 in mydict[line].keys():  
         temp = mydict[line][line2]  
         sum += int(temp[n])  
         count += 1

Q: Is this constant throughout the program?
We know what the keys and values represent.

```json
{ 'Fire': { 'Charmander': [309, 39, 52, 43, 60, 50, 100], 'Charmeleon': [405, 58, 64, 69, 65, 90, 80] }, 'Water': { 'Squirtle': [324, 44, 58, 65, 43, 78, 21], 'Wartortle': [405, 59, 53, 80, 65, 58, 40] } }
```
We know what the keys and values represent.

```
{ 'Fire': { 'Charmander': [309, 39, 52, 43, 60, 50, 100],
             'Charmeleon': [405, 58, 64, 69, 65, 90, 80]},
  'Water': { 'Squirtle': [324, 44, 58, 65, 43, 78, 21],
             'Wartortle': [405, 59, 53, 80, 65, 58, 40] } }
```
Use meaningful names.

pokemon types

{ 'Fire': { 'Charmander': [309, 39, 52, 43, 60, 50, 100],
            'Charmeleon': [405, 58, 64, 69, 65, 90, 80]},
  'Water': { 'Squirtle': [324, 44, 58, 65, 43, 78, 21],
             'Wartortle': [405, 59, 53, 80, 65, 58, 40]}}

pokemon names

pokemon stats
The number of pokemon stats (Total, HP, etc.) used does not change.

```json
{ 'Fire': { 'Charmander': [309, 39, 52, 43, 60, 50, 100], 'Charmeleon': [405, 58, 64, 69, 65, 90, 80]}, 'Water': { 'Squirtle': [324, 44, 58, 65, 43, 78, 21], 'Wartortle': [405, 59, 53, 80, 65, 58, 40] } }
```
Use constants for simple values that do not change.

```
{ 'Fire': { 'Charmander': [309, 39, 52, 43, 60, 50, 100],
            'Charmeleon': [405, 58, 64, 69, 65, 90, 80]},
 'Water': { 'Squirtle': [324, 44, 58, 65, 43, 78, 21],
            'Wartortle': [405, 59, 53, 80, 65, 58, 40]} }
```
avg_dict = {}
for line in mydict.keys():
    for n in range(7):
        sum = 0
        count = 0
        for line2 in mydict[line].keys():
            temp = mydict[line][line2]
            sum += int(temp[n])
        count += 1

Exercise: Rewrite using descriptive names and a constant.
avg_dict = {}
for poke_type in mydict.keys():
    for n in range(MAX_PSTATS):
        sum = 0
        count = 0
        for poke_name in mydict[poke_type].keys():
            poke_stats = mydict[poke_type][poke_name]
            sum += int(poke_stats[n])
            count += 1

Better: descriptive names and a constant
for key in rhym_dict:
    for i in range (len(rhym_dict[key])):
        for n in range (len(rhym_dict[key][i])):
            if rhym_dict[key][i][n][-1] == "1":
                prime_loc = n
                if rhym_dict[key][i][prime_loc:] == phonemes and \
                   rhym_dict[key][i][prime_loc-1] != precedes_p:
                    perfect_rhym.append(key)

Q: IS THIS GOOD STYLE?
for key in rhym_dict:
    for i in range (len(rhym_dict[key])):
        for n in range (len(rhym_dict[key][i])):
            if rhym_dict[key][i][n][-1] == "1":
                prime_loc = n
                if rhym_dict[key][i][prime_loc:] == phonemes and \rhym_dict[key][i][prime_loc-1] != precedes_p:
                    perfect_rhym.append(key)
for key in rhym_dict:
  for i in range (len(rhym_dict[key])):
    for n in range (len(rhym_dict[key][i])):
      if rhym_dict[key][i][n][-1] == "1":
        prime_loc = n
        if rhym_dict[key][i][prime_loc:] == phonemes and \rhym_dict[key][i][prime_loc-1] != precedes_p:
          perfect_rhym.append(key)

Unclear:  for i in range(...)  
Better:    for plist in rhym_dict[key]
for key in rhym_dict:
    for plist in rhym_dict[key]:
        for n in range(plist):
            if plist[n][-1] == "1":
                prime_loc = n
                if plist[prime_loc:] == phonemes and \
                    plist[prime_loc-1] != precedes_p:
                    perfect_rhym.append(key)
Summary

• Use continue (and break) carefully
  – avoid introducing complexity

• When operating on complex data structures:
  – use meaningful names for intermediate results to make the code easier to understand
  – use constants for clarity